

# FULL STACK DEVELOPMENT ESE3014

## Lab 8

Learning Outcomes:

1. Global Error Handling
2. Identity Layout and Logic
3. JavaScript Interop

## Lab Slide 28

### 1. Global Error Handling

Learning Outcomes:

Global Error Handling

#### Global Error Handling

##### *Practice 126*

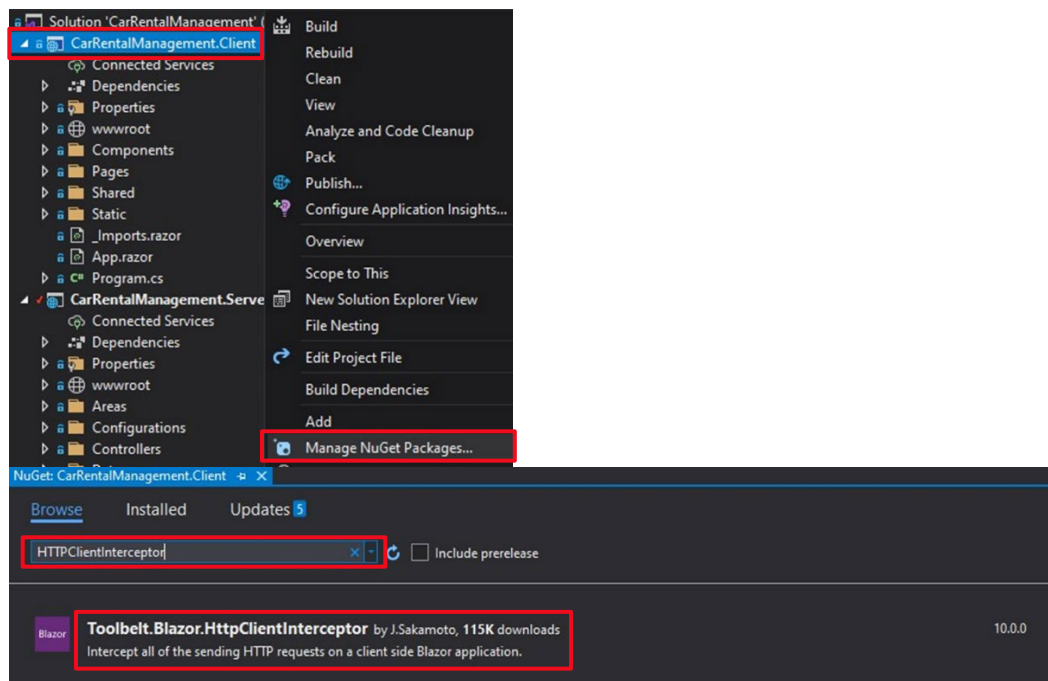
Handling errors in our web application can be very tedious if we use the standard Try...Catch method for every segment of code we have created. Instead of catching the error at the code level, it is possible to monitor the HTTP status of our web API calls.

The idea is: in the event of an error, before it can be reported to the browser, it will be intercepted and the error can be handled accordingly. Handling of errors can be in many forms like redirection to a page, logging the error via email, etc.

In the solution explorer, go to our client project. Right-click and select **Manage NuGet Packages...**

In the **NuGet Package Manager**, search for **HttpClientInterceptor**.

Install the **Toolbelt.Blazor.HttpClientInterceptor**



Open Program.cs file in the client project. Update the following code:

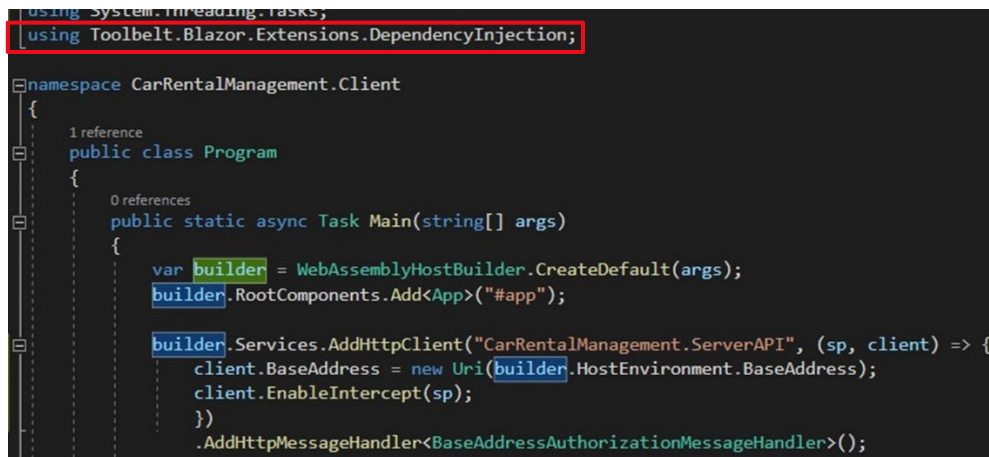
```
builder.Services.AddHttpClient("CarRentalManagement.ServerAPI",
client => client.BaseAddress = new
Uri(builder.HostEnvironment.BaseAddress))
.AddHttpMessageHandler<BaseAddressAuthorizationMessageHandler>();
```

The updated code is marked with red font:

```
builder.Services.AddHttpClient("CarRentalManagement.ServerAPI", (sp,
client) => {
    client.BaseAddress = new
Uri(builder.HostEnvironment.BaseAddress);
    client.EnableIntercept(sp);
})
.AddHttpMessageHandler<BaseAddressAuthorizationMessageHandler>();
```

Add the using statement:

```
using Toolbelt.Blazor.Extensions.DependencyInjection;
```



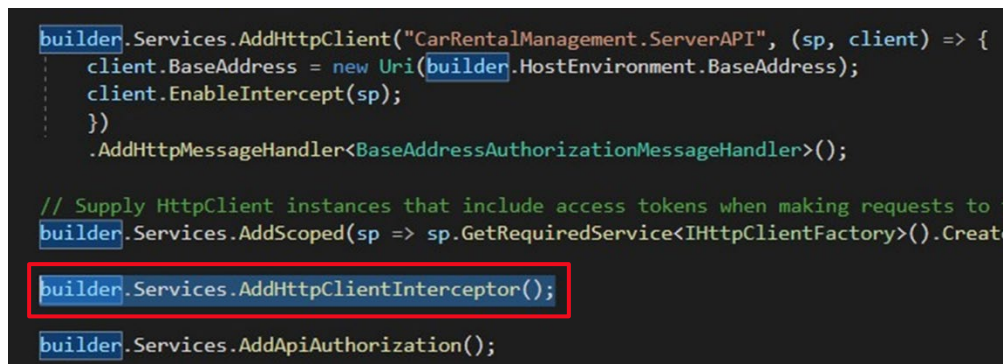
```
using System.Threading.Tasks;
using Toolbelt.Blazor.Extensions.DependencyInjection;

namespace CarRentalManagement.Client
{
    1 reference
    public class Program
    {
        0 references
        public static async Task Main(string[] args)
        {
            var builder = WebAssemblyHostBuilder.CreateDefault(args);
            builder.RootComponents.Add<App>("#app");

            builder.Services.AddHttpClient("CarRentalManagement.ServerAPI", (sp, client) => {
                client.BaseAddress = new Uri(builder.HostEnvironment.BaseAddress);
                client.EnableIntercept(sp);
            })
            .AddHttpMessageHandler<BaseAddressAuthorizationMessageHandler>();
        }
    }
}
```

Register the service by adding this line of code:

```
builder.Services.AddHttpClientInterceptor();
```



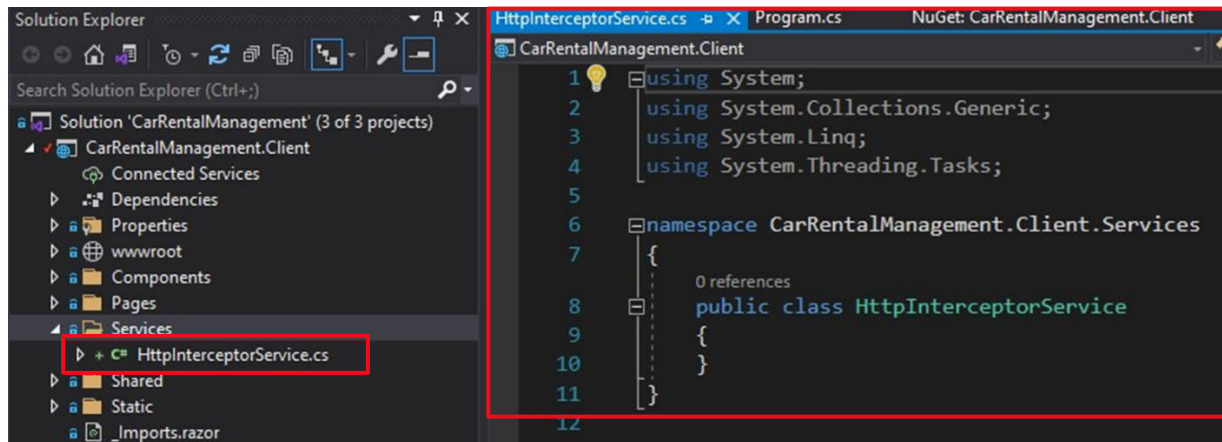
```
builder.Services.AddHttpClient("CarRentalManagement.ServerAPI", (sp, client) => {
    client.BaseAddress = new Uri(builder.HostEnvironment.BaseAddress);
    client.EnableIntercept(sp);
})
.AddHttpMessageHandler<BaseAddressAuthorizationMessageHandler>();

// Supply HttpClient instances that include access tokens when making requests to
builder.Services.AddScoped(sp => sp.GetRequiredService<IHttpClientFactory>().Create

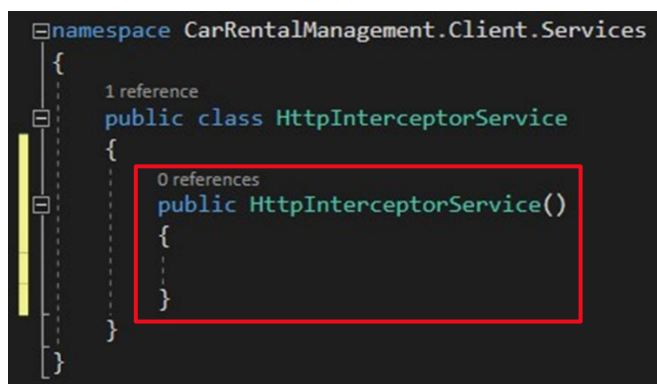
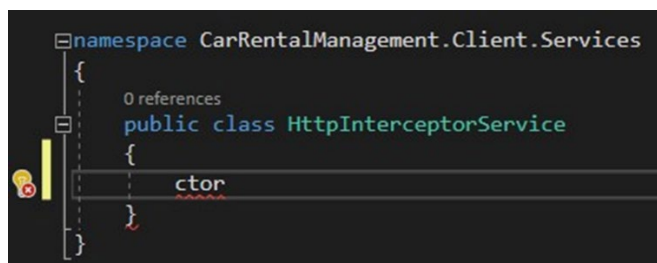
builder.Services.AddHttpClientInterceptor();

builder.Services.AddApiAuthorization();
```

In the client project, add a new folder named Services. Add a new class named HttpInterceptorService.cs



In the public class HttpInterceptorService method, add a constructor. Type ctor followed by pressing tab once. A constructor will be generated.



Add HttpClientInterceptor interceptor and NavigationManager navManager as parameters. Include the using references for both HttpClientInterceptor and NavigationManager.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6
7  namespace CarRentalManagement.Client.Services
8  {
9      1 reference
10     public class HttpInterceptorService
11     {
12         0 references
13         public HttpInterceptorService(HttpClientInterceptor interceptor, NavigationManager navManager)
14         {
15         }
16     }
17 }
    
```

using Toolbelt.Blazor;  
Toolbelt.Blazor.HttpClientInterceptor  
Generate type 'HttpClientInterceptor'  
Add null check  
Add null checks for all parameters  
Create and assign property 'interceptor'  
Create and assign field 'interceptor'  
Create and assign remaining as properties  
Create and assign remaining as fields

CS0246 The type or namespace name 'HttpClientInterceptor' could not be found (are you missing a using directive or an assembly reference?)  
Lines 4 to 6  
using System.Threading.Tasks;  
using Toolbelt.Blazor;  
Preview changes

Highlight the **interceptor** and click on the light bulb near the line number. Select **Create and assign field 'interceptor'**. Do not choose property as we required the field to be private readonly. Create and assign field for navManager.

```

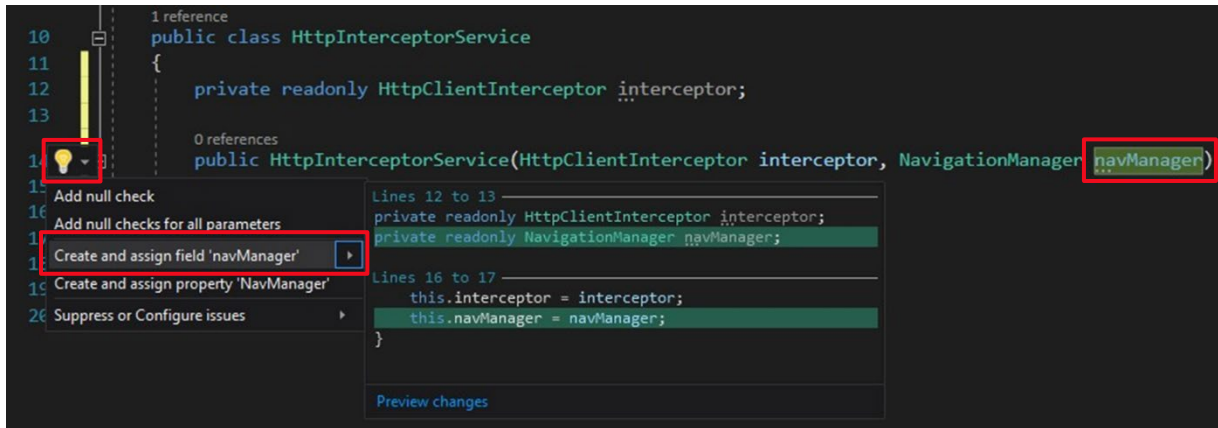
10 public class HttpInterceptorService
11 {
12     0 references
13     public HttpInterceptorService(HttpClientInterceptor interceptor, NavigationManager navManager)
14     {
15         private readonly HttpClientInterceptor interceptor;
16         public HttpInterceptorService(HttpClientInterceptor interceptor, NavigationManager navManager)
17         {
18             this.interceptor = interceptor;
19         }
20     }
21 }
    
```

Add null check  
Add null checks for all parameters  
Create and assign property 'interceptor'  
Create and assign field 'interceptor'  
Create and assign remaining as properties  
Create and assign remaining as fields  
Suppress or Configure issues

Lines 11 to 15  
private readonly HttpClientInterceptor interceptor;  
public HttpInterceptorService(HttpClientInterceptor interceptor, NavigationManager navManager)  
{  
 this.interceptor = interceptor;  
}

Preview changes

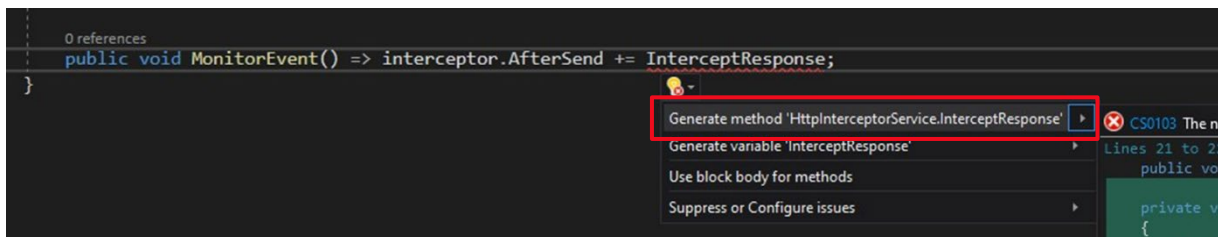
Highlight the **navManager** and click on the light bulb near the line number. Select **Create and assign field 'navManager'**. Do not choose property as we required the field to be private readonly.



Next, we will need to setup the event handlers. Add the following methods for event monitoring and event disposing.

```
public void MonitorEvent() => interceptor.AfterSend +=
InterceptResponse;
```

Select **Generate method 'HttpInterceptorService.InterceptResponse'**.



To implement the `InterceptResponse` method. We want to check if the response is not a success HTTP status code, then do something about it.

```

private void InterceptResponse(object sender,
HttpClientInterceptorEventArgs e)
{
    string message = string.Empty;
    if (!e.Response.IsSuccessStatusCode)
    {
        var responseCode = e.Response.StatusCode;

        switch (responseCode)
        {
            //To be implemented
        }
    }
}

```

```
}  
}
```

In the switch case. Here, we decide what to do if a certain HTTP status code is detected. Include the using reference for HttpStatusCode.

```
switch (responseCode)  
{  
    case HttpStatusCode.NotFound:  
        navManager.NavigateTo("/404");  
        message = "The requested resource was not found.";  
        break;  
    case HttpStatusCode.Unauthorized:  
    case HttpStatusCode.Forbidden:  
        navManager.NavigateTo("/unauthorized");  
        message = "You are not authorized to access this resource. ";  
        break;  
    default:  
        navManager.NavigateTo("/500");  
        message = "Something went wrong, please contact  
Administrator";  
        break;  
}
```

For event disposing, this method is to ensure that after monitoring the web API call, the event handler can be disposed. Add the following code:

```
public void DisposeEvent() => interceptor.AfterSend -=  
InterceptResponse;
```

Go back to Program.cs file and add the following line of code:

```
builder.Services.AddScoped<HttpInterceptorService>();
```

Remember to include the using reference for HttpInterceptorService.

```
builder.Services.AddHttpClientInterceptor();  
builder.Services.AddScoped<HttpInterceptorService>();  
builder.Services.AddApiAuthorization();
```



### Practice 127

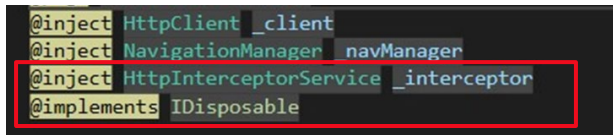
After setting up the `HttpInterceptorService`, it is time to inject it to the pages.

Go to `Colours` folder, open `Index.razor` file. Add the following code:

```
@inject HttpInterceptorService _interceptor
@implements IDisposable
```

Remember to include the using reference for `HttpInterceptorService` in the **Imports.razor** file.

```
@using CarRentalManagement.Client.Services
```

A screenshot of a code editor showing the injection of `HttpInterceptorService` into the `Index.razor` file. The code is: 

```
@inject HttpClient _client
@inject NavigationManager navManager
@inject HttpInterceptorService _interceptor
@implements IDisposable
```

 The third line, `@inject HttpInterceptorService _interceptor`, and the fourth line, `@implements IDisposable`, are highlighted with a red rectangular box.

Add the dispose method and set up the monitoring event.

A screenshot of a code editor showing the `OnInitializedAsync` and `Dispose` methods in the `Index.razor` file. The `OnInitializedAsync` method is defined as 

```
protected async override Task OnInitializedAsync()
{
    _interceptor.MonitorEvent();
    Colours = await _client.GetFromJsonAsync<List<Colour>>($"${Endpoints.ColoursEndpoint}");
}
```

 The line `_interceptor.MonitorEvent();` is highlighted with a red rectangular box. The `Dispose` method is defined as 

```
public void Dispose()
{
    _interceptor.DisposeEvent();
}
```

 The entire `Dispose` method block is highlighted with a red rectangular box.



### Practice 128

In the Pages folder, create the error pages: 404.razor, 500.razor and Unauthorized.razor. Right-click on the Pages folder and add Razor Component.

For 404.razor:

```
@page "/404"

<h1 class="text-danger">
    The resource you are looking for could not be found.
</h1>
```

For 500.razor:

```
@page "/500"

<h1 class="text-danger">
    We have encountered an error completing your request. Please try
    again.</h1>
```

For Unauthorized.razor:

```
@page "/unauthorized"

<h1 class="text-danger">
    You are not authorized to access this resource.
</h1>
```

### Practice 129

To test, we need to simulate an error. Go to Server project, in Controllers folder, open the ColoursController.cs file. At the GET API method, add

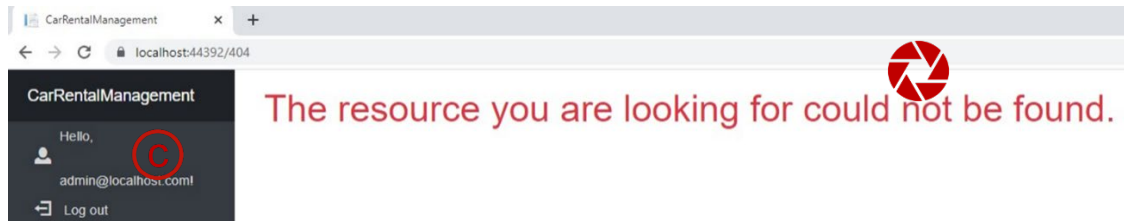
```
return NotFound();
```

This code is only for testing and it should be removed after testing.

```
// GET: api/Colours
[HttpGet]
//Refactored
//public async Task<ActionResult<IEnumerable<Colour>>> GetColours()
0 references
public async Task<ActionResult> GetColours()
{
    //To be deleted or comment after testing the Global Error Handling
    return NotFound();
    //Refactored
    //return await _context.Colours.ToListAsync();
    var Colours = await _unitOfWork.Colours.GetAll();
    return Ok(Colours);
}
```

**Exercise 43**

- Run the application and log-in.
- Select Colour in the NavMenu.
- Submit the full screenshot of the blazor index page for Colour.



## Lab Slide 29

## 2. Identity Layout and Logic

Learning Outcomes:

Identity Layout and Logic

- i) Identity Layout and Logic
- ii) The Log Out Fix

---

### Identity Layout and Logic

#### *Practice 130*

In this project, we have scaffolded all the default pages relating to the .NET Identity. If you have not done so, visit the earlier labs for instructions. You select the required pages and add them to the **Areas** folder in **Server project**.

Go to the **Areas** folder, followed by the **Identity** folder, the **Pages** folder and finally in the **Account** folder, you will find some familiar pages like the **Login.cshtml**, **Register.cshtml**, etc.

When user access pages that requires authorization, the user will be redirected to the login page to authenticate their credentials.

The page design is basic and the developer can choose to edit it to suit the web application theme.

Let's modify the login page. First, remove the layer that contains Use another service to log in. Open the **Login.cshtml** file and look for line 49. Collapse the div tag and delete the div layer. Second, expand the Log in form layer. Go to line 10, change the code `<div class="col-md-4">` to `<div class="col-md-12">`.

Modify the **Register.cshtml** to standardize the look and feel.

CarRentalManagement.Server

Log in

Use a local account to log in.

Email

Password

☐ Remember me?

Log in

[Forgot your password?](#)  
[Register as a new user](#)  
[Resend email confirmation](#)

Use another service to log in.

There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.

CarRentalManagement.Server

Register Login

Log in

Use a local account to log in.

Email

Password

☐ Remember me?

Log in

[Forgot your password?](#)  
[Register as a new user](#)  
[Resend email confirmation](#)

© 2021 - CarRentalManagement.Server - Privacy

To find the file that sets the global layout, open **\_ViewStart.cshtml**. The layout path points to **"/Pages/Shared/\_Layout.cshtml"**. From **Server** project, go to **Pages** folder, followed by **Shared** folder. Open the **\_Layout.cshtml** file.

Let's change the application titles.

Look for **line 10** and modify the code to:

```
<title>@ViewData["Title"] - Car Rental Management Application</title>
```

Look for **line 28** and modify the code to:

```
<a class="navbar-brand" href="~/>Car Rental Management Application</a>
```

Look for **line 60** and modify the code to:

```
&copy; 2023 - Car Rental Management Application
```

The modified layout will display accordingly.

Car Rental Management Application

Register Login

Log in

Use a local account to log in.

Email

Password

☐ Remember me?

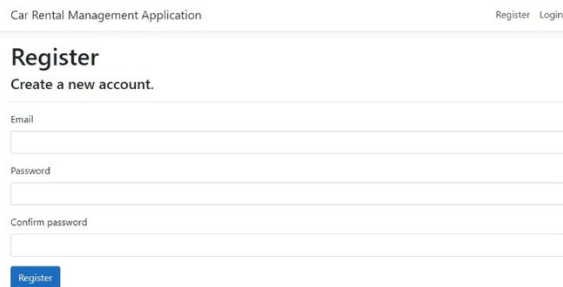
Log in

[Forgot your password?](#)  
[Register as a new user](#)  
[Resend email confirmation](#)

© 2021 - Car Rental Management Application - Privacy

### Practice 131

Previously in the ApplicationUser.cs file, we have extended the AspNetUser table in the database to store first and last name. Starting from the Register page, let's modify the registration form to accept first and last name. At the same time, we will put in code logic to assign new user to the User role using .NET Identity.



Open Register.cshtml.cs file. This is a code-behind file for Register.cshtml. Find the InputModel class and add the FirstName and LastName properties.

```
[Required]
[Display(Name = "First Name")]
public string FirstName { get; set; }

[Required]
[Display(Name = "Last Name")]
public string LastName { get; set; }
```

```
1 reference
public class InputModel
{
    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    7 references
    public string Email { get; set; }

    [Required]
    [Display(Name = "First Name")]
    4 references
    public string FirstName { get; set; }

    [Required]
    [Display(Name = "Last Name")]
    4 references
    public string LastName { get; set; }
```

Further down the file, find OnPostAsync method and modify the code accordingly. There are two parts, first the creation of the user and second the assignment of User role.

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).To
    if (ModelState.IsValid)
    {
        var user = CreateUser();

        await _userManager.SetUserNameAsync(user, Input.Email, CancellationToken.None);
        await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);

        //Set user first and last name
        user.FirstName = Input.FirstName;
        user.LastName = Input.LastName;

        var result = await _userManager.CreateAsync(user, Input.Password);

        if (result.Succeeded)
        {
            //Set user role to user
            if (!await _roleManager.RoleExistsAsync("User"))
            {
                await _roleManager.CreateAsync(new IdentityRole("User"));
            }
            await _userManager.AddToRoleAsync(user, "User");

            _logger.LogInformation("User created a new account with password.");
        }
    }
}
```

Go to the top of the file, add the RoleManager field and populate the constructor. Remember to include the using reference for **RoleManager**.

```
[AllowAnonymous]
8 references
public class RegisterModel : PageModel
{
    private readonly SignInManager<ApplicationUser> _signInManager;
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly ILogger<RegisterModel> _logger;
    private readonly IEmailSender _emailSender;
    private readonly RoleManager<IdentityRole> _roleManager;

    0 references
    public RegisterModel(
        UserManager<ApplicationUser> userManager,
        SignInManager<ApplicationUser> signInManager,
        ILogger<RegisterModel> logger,
        IEmailSender emailSender,
        RoleManager<IdentityRole> roleManager)
    {
        _userManager = userManager;
        _signInManager = signInManager;
        _logger = logger;
        emailSender = emailSender;
        _roleManager = roleManager;
    }
}
```

Open the Startup.cs file, find services.AddDefaultIdentity and modify the code.

```
services.AddDefaultIdentity<ApplicationUser>(options =>
options.SignIn.RequireConfirmedAccount = false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Remember to include the using reference for **IdentityRole**.

```
services.AddDefaultIdentity<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

The logic is completed but we are still not done yet. Now, moving on to the user interface. Open the Register.cshtml file. Add the form controls for FirstName and LastName.

```
<div class="row">
  <div class="col-md-4">
    <form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl" method="post">
      <h2>Create a new account.</h2>
      <hr />
      <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
      <div class="form-floating mb-3">
        <input asp-for="Input.Email" class="form-control" autocomplete="username" aria-required="true" placeholder="name@example.com" />
        <label asp-for="Input.Email">Email</label>
        <span asp-validation-for="Input.Email" class="text-danger"></span>
      </div>
      <div class="form-floating mb-3">
        <input asp-for="Input.FirstName" class="form-control" autocomplete="give-name" aria-required="true" />
        <label asp-for="Input.FirstName">First Name</label>
        <span asp-validation-for="Input.FirstName" class="text-danger"></span>
      </div>
      <div class="form-floating mb-3">
        <input asp-for="Input.LastName" class="form-control" autocomplete="family-name" aria-required="true" />
        <label asp-for="Input.LastName">Last Name</label>
        <span asp-validation-for="Input.LastName" class="text-danger"></span>
      </div>
      <div class="form-floating mb-3">
        <input asp-for="Input.Password" class="form-control" autocomplete="new-password" aria-required="true" placeholder="password" />
```



### Exercise 44

- Run the application and register a new user.
- Submit the full screenshot of the registration page and the index page after registration.

Car Rental Management Application

## Register

Create a new account.


Email

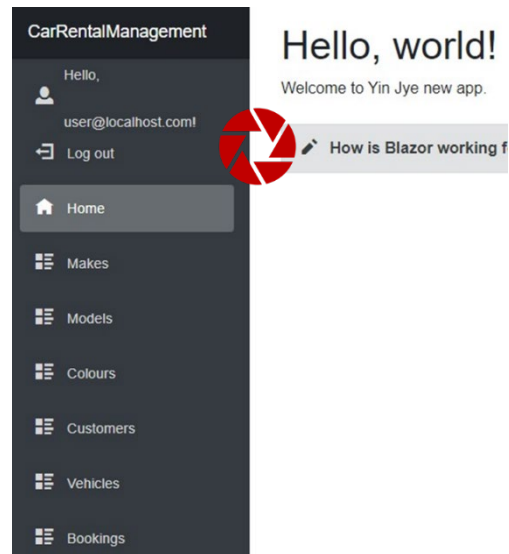
First Name

Last Name

Password

Confirm password





### 3. JavaScript Interop

Learning Outcomes:

JavaScript Interop

i) JavaScript Interop – DataTables

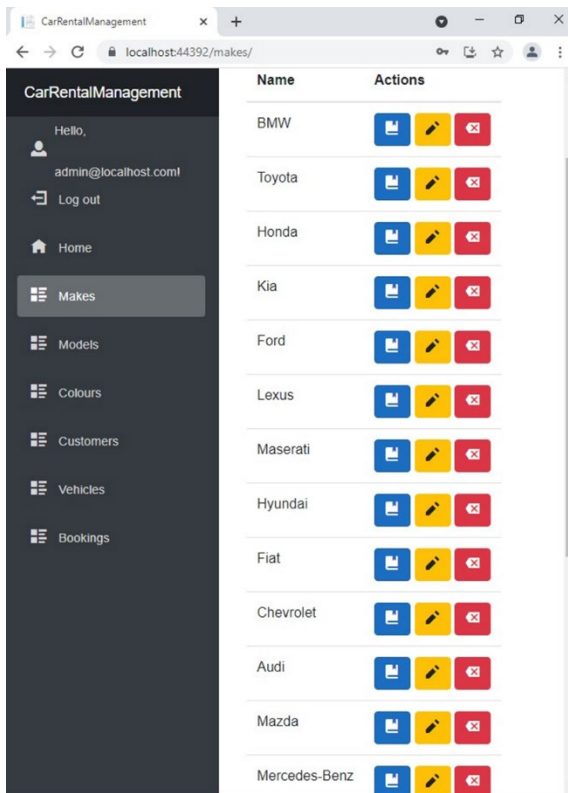
---

#### JavaScript Interop

##### *Practice 132*

The blazor development provides the default application UI based on Bootstrap CSS Framework. It is common for application owners to request for a different interface rather than using the default.

In the Makes index page, you may notice that if you keep on adding Make data, the rows will accumulate and soon you will have a very long list. This is not so desirable in comparison to today's expectation of UI design.



Let's use a third-party library to help make this list of data better. We will be using **DataTables**. You can find the resource and documentation link here:

<https://datatables.net/examples/styling/bootstrap5.html>

DataTables provide additional functionalities like search, paging, sorting, etc. Using DataTables as an example to provide insight to how blazor interacts with JavaScript.

In the resource link, you can find the JavaScript files that we will need to include in our project to make this work as well as the code to initialize the DataTables in our blazor pages.

JavaScript

HTML

CSS

Comments (0)

The Javascript shown below is used to initialise the table shown in this example:

```
1 | new DataTable('#example');
```

In addition to the above code, the following Javascript library files are loaded for use in this example:

- <https://code.jquery.com/jquery-3.7.0.js>
- <https://cdn.datatables.net/1.13.7/js/jquery.dataTables.min.js>
- <https://cdn.datatables.net/1.13.7/js/dataTables.bootstrap5.min.js>

The developer can decide to import these JavaScript files as links or as physical files. In our application, let import them as links. From the Client project, open the **index.html** found in the **wwwroot** folder.

It is very important that you import the links before this line of code.

```
<script src="_framework/blazor.webassembly.js"></script>
```

```
<script src="https://code.jquery.com/jquery-3.7.0.js"></script>
<script src="https://cdn.datatables.net/1.13.7/js/jquery.dataTables.min.js"></script>
<script src="https://cdn.datatables.net/1.13.7/js/dataTables.bootstrap5.min.js"></script>

<script src="_content/Microsoft.AspNetCore.Components.WebAssembly.Authentication/AuthenticationService.js"></script>
<script src="_framework/blazor.webassembly.js"></script>
<script>navigator.serviceWorker.register('service-worker.js');</script>
```

Go back to the resource link and find the link for CSS file. You will just require the DataTables CSS file. The other file is the Bootstrap CSS file which is already imported when you first create your Blazor application.

JavaScript

HTML

CSS

Comments (0)

The following CSS library files are loaded for use in this example to provide the styling of the table:

- <https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/5.3.0/css/bootstrap.min.css>
- <https://cdn.datatables.net/1.13.7/css/dataTables.bootstrap5.min.css>

```
<link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
<link href="css/app.css" rel="stylesheet" />
<link href="https://cdn.datatables.net/1.13.7/css/dataTables.bootstrap5.min.css" rel="stylesheet" />
<link rel="icon" type="image/png" href="favicon.png" />
```

Next, write a script file to dynamically use the DataTables. Add a new folder named **scripts** in the **wwwroot** folder. Add a new item, select a JavaScript File, named it as **datatable.js** and add the following JavaScript functions:

```
function AddDataTable(table) {
    $(document).ready(function () {
        $(table).DataTable();
    })
}

function DataTablesDispose(table) {
    $(document).ready(function () {
        $(table).DataTable().destroy();
        var element = document.querySelector(table + '_wrapper');
        element.parentNode.removeChild(element);
    })
}
```

Once the **datatable.js** is created, import this JavaScript file in the **index.html**.

```
<script src="scripts/datatable.js"></script>
```

```
<script src="https://code.jquery.com/jquery-3.7.0.js"></script>
<script src="https://cdn.datatables.net/1.13.7/js/jquery.dataTables.min.js"></script>
<script src="https://cdn.datatables.net/1.13.7/js/dataTables.bootstrap5.min.js"></script>
<script src="scripts/datatable.js"></script>

<script src="_content/Microsoft.AspNetCore.Components.WebAssembly.Authentication/Authentication/Authentication.js"></script>
<script src="_framework/blazor.webassembly.js"></script>
<script>navigator.serviceWorker.register('service-worker.js');</script>
```

In the Client project, go to Bookings folder and open Index.razor file.

Add `@implements IDisposable`

```
@page "/bookings/"
@inject HttpClient _client
@inject IJSRuntime js
@attribute [Authorize]
@implements IDisposable

<h3 class="card-title">Car Bookings</h3>
<br />
```

Modify the `<table>` tag and add id attribute.

```
<table class="table table-responsive" id="bookingsTable">
```

```
@if (Bookings == null)
{
    <div class="alert alert-info">Loading Bookings...</div>
}
else
{
    <table class="table table-responsive" id="bookingsTable">
        <thead>
```

In the code block, add these two methods. The `OnAfterRenderAsync` override method execute after all the blazor components have been rendered. This will allow the JavaScript execute safely afterwards. The `Dispose` method destroys the `DataTable` each time the page is being disposed of. This removes any existing render of the `DataTable` before the next creation of a new `DataTable`.

```
protected async override Task OnAfterRenderAsync(bool firstRender)
{
    await js.InvokeVoidAsync("AddDataTable", "#bookingsTable");
}

public void Dispose()
{
    js.InvokeVoidAsync("DataTablesDispose", "#bookingsTable");
}
```







**Exercise 45**

- Run the application, login first and load the Bookings page.
- Submit the full screenshot of the Bookings Index page.

**Car Bookings**

[+ Create New Booking](#)

Show  entries Search:

Booking Id	Customer License	Date	Duration in days	Plate Number	Actions
1	S7654321Z	03 January 2022	5	SMC1234A	  
2	S1234567A	29 December 2021	To Be Determined	SGG1234B	  

Showing 1 to 2 of 2 entries Previous **1** Next